Automated Support for Mathematical Datatypes via Trigger-based Axiomatization

Prasoon Dadhich,
Master Parisien de Recherche en Informatique
work done at Chair of Programming Methodology ETH Zurich
under the guidance of
Dr. Alexander J. Summers
ETH Zurich
Prof. Peter Müller
ETH Zurich

21^{st} August 2015

The general context

Satisfiability Modulo Theories solvers are the core of several verification technologies. Verification tasks such as checking verification conditions in deductive verification or computing program abstraction in software model checking can be reduced to the first order theories and solved in SMT solvers. SMT solvers have native support for various theories such as linear arithmetic, arrays and bit vectors. Some verification tasks involve reasoning of other complex theories such as sequences which are not natively supported. Adding a new theory to SMT solvers depends on implementation details of the SMT solver, and is done mainly by the developer. These theories can be alternatively supported by first order axiomatization. However, in the presence of quantifiers, SMT solvers are incomplete and exhibit unpredictable behaviour such as non-termination.

The research problem

Triggers or patterns can be used to handle quantifiers in the first order axiomatization. However, an uncontrolled instantiation with triggers or patterns can give non-terminating and incomplete behaviour. Therefore, if the trigger-based axiomatization are not designed carefully, they give an unpredictable behaviour to the user and makes it hard to debug. The axiomatization can also lead to non termination because of wrong order in instantiating the trigger or a matching loop. In the pursuit of completeness, sometimes the axioms are instantiated exponentially which gives several duplicate axioms. This arises performance issue in the solver for goals that originate from the verification programs manipulating these data structures. Our goal is to study further about the problem of proving completeness and termination with also keeping in mind about the performance issue.

The theory of mathematical sequences is one theory which is widely used in verification tools for automatic support. It provides an excellent platform to practice proofs by induction and also serves

as a basis of reasoning about lists in the verification language such as why3, silver etc. However, due to several universal quantifiers and complexity of the theory, its difficult to come up with a good trigger-based axiomatization.

Proposed contribution

There is no general recipe for designing a complete and terminating trigger-based axiomatization which can work for all kind of theories. However, we investigated and proposed a number of general techniques and guidelines which we found useful to design a sound, complete and terminating axiomatization. We designed a trigger-based first order axiomatization for Sequence theory, proved and argued on its soundness, completeness and termination with respect to the triggers used for instantiation. We also show a general way to avoid duplicate instantiations of axioms with so-called "stub-functions" which improves the performance significantly.

Arguments supporting its validity

We have the proofs and arguments for soundness, termination and completeness of the Sequence axiomatization using several techniques and formalization. The new set of sequences axioms with triggers is experimented with various verification tools and have shown significant performance improvement in the verification tools.

Summary and future work

SMT solvers with user provided axiomatization performs badly while handling quantifier instantiations. We could solve this problem to some extent for particular set of axioms using stub functions. But there are still problems related to performance in triggered axiomatization which can be addressed if investigated deeply over the instantiation tree. With respect to the sequence axiomatization, one can furthermore think of providing a robust solution to the support of extensionality completely.

There has been significant work done in formalization but I think investigating furthermore about the general way of implementing a good axiomatization can be fruitful for various verification tools.

Notes and acknowledgement

I wish to thank my supervisors Prof. Peter Müller and Dr. Alexander J. Summers for their genuine help and keen insights which made this research internship fruitful.

The report is written in English, because the author and supervisors know very little French.

1 Background

1.1 Axiomatization

Satisifiability Modulo Theories (SMT) solvers decide satisfiability or unsatisfiability of first order formulas in the presence of background theories such as booleans, integer-arithmetic, bit-vectors, arrays etc. However, in general, verification tools need support for various complex theories. as the verification conditions generated should be evaluated modulo a theory describing the verification methodology. SMT solvers cannot support such arbitrary theories natively. Given the complexity of such theories, it seems highly impractical to implement inside an SMT solver. Therefore, a deductive first order axiomatization is typically developed using the theories available in the solver natively. An axiomatization to support a theory consists of universally quantified formulas. SMT solvers can use several techniques. Model-based quantifier instantiation and trigger based quantifier instantiations are most widely used; our work concentrates on the latter.

Triggers The Stanford Pascal verifier and the subsequent Simplify theorem prover [9] pioneered the use of trigger-based quantifier instantiation. The basic idea behind trigger-based quantifier instantiation is quite straight-forward. Firstly, annotate a quantified formula using a pattern or trigger that contains all the bound variables. A pattern is an expression (that does not contain binding operations, such as quantifiers) that contains variables bound by a quantifier. Then instantiate the quantifier whenever a term that matches the pattern is created during proof search. This is a conceptually simple starting point, but there are several subtleties that are important.

We show an example below to describe triggers in universally quantified axioms:

$$\forall x : int\ [g(x)]$$
$$\text{f(g(x),0)=0}$$

In the axiom f and g are function symbols, and x is a quantified integer variable. The square bracket shows the trigger or the pattern. In the above example, taking all terms of integer type to instantiate axiom is too much. With the help of the trigger $g(x)$ we do not allow instantiation with all such terms. Instead, we instantiate x only with terms t for which a ground term $g(t)$ exists. This trigger can be seen as a heuristic which can be given by the user or computed by the solver.

1.2 Motivation

Quantifier handling in trigger-based first order axiomatization are generally seen unreliable with no termination and completeness guarantees. Most of the time a wrongly chosen trigger with no proper reasoning can diverge the instantiations. The performance issue is also a problem when duplicate axioms are instantiated. The E-matching algorithm [17] which is a well known approach for quantifier reasoning also has some limitations [8]. It needs ground seeds terms, otherwise it fails to prove simple properties when ground terms are not available. It also fails to detect the diverging triggering because of the two mutually recursive triggers. This is generally termed as matching loop. A small example to show matching loop:

$$\forall x\ [f(x)] \quad f(x) = g(f(x)))$$

$$\forall x\ [g(x)] \quad g(x) = f(g(x)))$$

In above axiom, if we start with a ground term $f(x)$, the trigger in the left axiom $f(x)$ will unlock a new trigger of shape $g(f(x))$ which will trigger the right axiom and unlock a new term of shape $f(g(f(x))$ and this mutual triggering process will repeat infinitely leading to non-termination. Therefore, these limitations are to be dealt with care while working on a trigger based axiomatization. There has been considerable work done in past to show that an axiomatization with correctly chosen triggers, if proven sound, complete and terminating, will work as a generic decision procedure [11] . As there is no universal recipe to build a new axiomatization for a theory, it requires a lot of reasoning with triggers and theory specific knowledge to prove completeness and termination.

Sequences is one of the most widely used theory as a first order axiomatization in various verification infrastructures such as Why3 [4] , Dafny [15] and Viper [14]. It gives a good platform to practice proof by induction. However, due to its complexity involved in instantiating universally quantified axioms, leads to several unpredictable behaviours from the solver. This is mainly because the triggered axiomatizations have not been studied in detail with respect to completeness and termination. Therefore, investigating a better general way of designing the correct triggered axiomatization such as sequences is potentially very useful for the verification tools.

1.3 Goal results

The goal of the thesis was to give a sound, terminating and complete first order logic axiomatization for the theories of mathematical sets and sequences without compromising with the performance of the tool. In order to solve this problem, we also came up with general techniques in addition to the techniques shown in [10]. which can be used as a guideline to design such axiomatizations for other theories.

2 Preliminaries

In this section we provide the definition of structures used throughout the thesis. At the end we define the background problem and the problem studied.

2.1 Theorem Provers and SMT solvers

In a verification infrastructure, the Verification conditions from programs are fed either in interactive theorem provers, automatic theorem provers or Satisfiability Modulo Theory solvers. We are interested in the automated program verification therefore we can consider ATPs and SMT solvers.

Vampire [19] is one of the stable automatic theorem prover which uses resolution and superposition to decide the satisfiability of first order propositional calculus with equality. Whereas, SMT solvers decide the satisfiability of a set of formulas with ground terms modulo a background theory. Different SMT solvers have native support for various background theories. Some of the most commonly supported are Equality and Uninterpreted Function (EUF), Linear Arithmetic (LIA etc.), Bit-Vectors and Arrays.

The SMT solver's core is the SAT solver which is generally based on Davis-Putnam-Logemann-Loveland (DPLL) algorithm.

Most SMT solvers provide native support for multiple background theories and it is often the case that we need the combination of them to solve a problem. There are several methods used to combine theories. The most common one is the Nelson-Oppen Combination method [6].

There are several SMT solvers developed. The Z3 SMT solver [7] developed at Microsoft Research is the default solver used by Boogie. It have several theories supported. The open source CVC4 SMT solver [2] is also very popular in academia. They too support several theories and recently have come up with a new efficient approach for handling conflicts in quantifier instantiation techniques [18].

The verification condition generated from our intermediate verification language Silver in VIPER tool infrastructure are discharged into Z3 solver.

2.2 Formalization

The formalization of the first order logic with a notation for triggers that restricts the quantifier instantiation was shown by Claire Dross in paper "Reasoning with Triggers" [11] . In this logic we then define properties- Soundeness, completeness and termination for the sequence axiomatization. We also give the performance improvement tricks for the recursive axioms with our axiomatization.

2.2.1 First Order Logic with Triggers and witnesses (FOL*)

Two new kind of formulas are introduced. A formula ψ under trigger l is written $[l]\psi$. It can be read as if a literal l is true and all its sub-terms are known then assume ψ. A dual construct for $[l]\psi$, which we call witness $\langle l\rangle\psi$. It can be read as assume that the literal l is true and all its sub-terms are known and assume ψ.

The extended syntax of formulas with First order logic are summarized as follows:

$$\psi ::= A \mid \psi_1 \vee \psi_2 \mid \psi_1 \wedge \psi_2 \mid \forall x.\psi \mid \exists x.\psi \mid \neg\psi \mid [l]\psi \mid \langle l\rangle\psi$$

2.2.2 Denotational Semantics of FOL*

The semantics of the FOL* language are defined via two encodings $[\![]\!]^+$ and $[\![]\!]^-$ into First order language. The notation $[\![]\!]^\pm$ is used when the rules are same for both the polarities and the polarities of the sub-formula does not change. A new unary predicate symbol *Known* is introduced which denotes the fact that a term is known. Given a term t or an atomic formula A, the set of all the non-variable sub-terms of t is denoted with $\mathcal{T}(t)$. The expression $Known(\mathcal{T}(t))$ stands for the conjunction $\wedge_{t'\in\mathcal{T}(t)} Known(t')$.

The entailment in FOL* is defined as follows

$$F \vdash^* \triangleq known(\omega), [\![F]\!]^- \vdash [\![G]\!]^+$$

where ω is an arbitrary fresh constant supposed to be known a priori, and $\vdash$ stands for entailment in FOL.

Definition 2.1. World modulo T : We call world a *T-satisfiable* set of ground literals. A world L is inhabited if there is at least one term occurring in it, i.e. $\mathcal{T}(L)$ is non-empty. A world L is complete if for every ground literal l in the signature of T, either $l \in L$ of $\neg l \in L$.

Definition 2.2. *Known* term modulo T : A term t is known in a world L if and only if there is a term t'.

2.3 Equational axioms and E-matching algorithm

During the proof search in an SMT solver, axioms are triggered by matching sub-expressions in the goal. In this triggered axiomatization, if we reason at the higher level, the axioms are all equations of this form

$$\forall x(t_{lhs} = t_{rhs})$$

where $FV\{t_{lhs}\} = x$ and $FV\{t_{rhs}\} \subseteq x$. The left-hand side t_{lhs} is generally the pattern or trigger.

While in practice, SMT-solvers support various kind of patterns in general, in this thesis, we stick to the pattern or trigger to be always on its left-hand side in the equational axioms.

Simplify [9] is a legacy SMT system, the first one to have efficiently combine theory and quantifier reasoning. Their technical report describes a recursive matching algorithm which has evolved into the concept of E-matching and is used in various SMT solvers. The high level idea behind the E-matching is as follows. The axiom is triggered by the activate trigger ψ of the solver, if ψ contains a subterm u and there exists a substitution θ such that $u =_E t_{lhs}\theta$, i.e. u matches the trigger of the axiom modulo Equality. If the axiom is triggered, then the current active term is replaced by the logically equivalent formula where u will be replaced by $t_{rhs}\theta$.
Therefore, the axioms which will be used in the axiomatization can finally be viewed as the rewriting rules as also described above in the formal semantics. Each application of an axiom will preserve the logical equivalence to the original goal.
As long as there exists an axiom with the active terms that can be triggered, then the triggering is guaranteed. Same term cannot be triggered again unless there is another axiom with mutually recursive trigger. This is why, the termination in general is not guaranteed for the mutually recursive axioms because they keep unlocking new triggers. Note that, unlike in term rewrite systems, there is no notion of term orderings or well-defined customizable strategies which could be used to guide the triggering process of the axioms. Therefore, sometimes while triggering, if the axiomatization leads to wrong ordering, it can lead to non-termination or longer path. The ordering can also instantiate duplicate axioms in some cases.

2.4 Concept of local theory extension

Local Theory A theory extension extends a given theory with additional symbols and axioms. Local theory extensions are class of such axioms that can be decided using finite quantifier instantiation of the extension axioms.

The triggered axiomatization can be further seen as a local theory extension [1, 13] which extends on a given base theory with additional symbols and axioms. We consider extensions $T_0 \bigcup \mathcal{K}$ with additional symbols and axioms satisfying a set $\mathcal{K}$. An extension $T_0 \subseteq T_0 \bigcup \mathcal{K}$ is local if satisfiability of a set G of clauses w.r.t. $T_0 \bigcup \mathcal{K}$ only depends on T_0 and those instances $\mathcal{K}[G]$ of $\mathcal{K}$ in which the terms starting with extension functions are in the set $st(\mathcal{K}, G)$ of ground terms which already occur in G or

$\mathcal{K}$. It allows to restrict the search to the instances $\mathcal{K}^{[g]}$ of $\mathcal{K}$ in which the variables below are extension functions are instantiated with σ_0-terms generated from $st(\mathcal{K}, G)$. But in the triggered axiomatization the local instances $\mathcal{K}$ with respect to the ground terms set $st(\mathcal{K}, G)$ are sometimes insufficient to yield a saitisfiability modulo the theory extensions.

3 Designing Trigger-based Axiomatization

There is no general way to design an axiomatization and nor to prove termination and completeness. The axiomatization and proofs are dependent on the theory we want to decide. In this section, we give general practices and our experiences, including debugging techniques, which can be applicable for several other theories with triggered axiomatizations.

3.1 Termination

There can be no single "true" definition of a terminating axiomatization. Different variations of solver with different algorithm may terminate on different classes of problems, which is difficult to describe and to reason about. In [10], description for these definitions are detailed.

To reason about termination in general, we need an abstract representation of the solver's state. It is convenient to see this evolution as game where we choose several universally quantified formulas to instantiate and our set of truth assignment decides how to interpret the result of the instantiation, that is what new facts can be assumed. Whenever we arrive at a set of facts that is inconsistent or saturated so that no new instantiations can occur, the game terminates and we win. If on the other hand, whatever instantiations we do, the assignments can find new universal formulas to instantiate, the game continues indefinitely. The axiomatization will be terminating if we have the winning strategy. The winning strategy in this context can be summarized as: no matter what partial order we explore, there will be a sequence of instantiations due to unlocked triggers which leads either to a conflict or a saturated partial model.

According to our experience, in order to design such a terminating axiomatization, one could classify the set of axioms in such a way that they can be reasoned for termination independently. Once you have this classification, we can look for the dependencies with terms between each classified sets of axioms. And in general, the axioms should unlock triggers of "smaller" size , in order to avoid diverging triggering.

Avoiding Matching Loops As mentioned earlier, E-matching does not prevent the possible non-termination which could be because of the two mutually recursive axioms or triggering with the wrong order leading to matching loop. This problem can be mitigated by choosing triggers that are larger than the new terms produced by the instantiations or converge the instantiation to the empty (halting) symbol. We give a small example to explain this phenomena.

Lets have a recursive Axiom *Append_ Append_ Associativity* which defines an associativity property quantified over three variables of polymorphic type $Seq\,T$.

$$\forall\ s_0,\ s_1,\ s_2 : Seq\,T\ \ [Append(s_0, Append(s_1, s_2))]$$
$$Append(s_0, Append(s_1, s_2)) \approx Append(Append(s_0, s_1), s_2)$$

If the axiom were triggered on $Append(Append(s_0, s_1), s_2)$ instead of $Append(s_0, Append(s_1, s_2))$, the triggering would have led to a matching loop. This is because of the wrong order chosen for triggering as in the first case the SMT solver would know the empty sequence case to stop instantiations whereas its not possible with the second trigger.

Performance optimization As mentioned in the E-matching, unlike in term rewriting systems, there is no notion of term orderings or well-defined customizable strategies which can be used to guide the triggering instances in the right direction. Therefore, sometimes triggering, leads to wrong ordering and diverges causing non-termination. Such a mistake can be possibly avoided by choosing the right trigger as shown earlier. But, sometimes, the triggering can also lead to instantiations of several duplicate instantiations which worsens the performance of the solver. We propose a general way of guiding the instantiations which works for all the recursive functions. We take the same recursive axiom *Append_ Append_ Associativity* to explain this technique. The axiom is self triggering or recursive, which instantiate several duplicate axioms and eventually gives us the same information. For example: In *Append_ Append_ Associativity* axiom, for the term which is nested with more than 2 $Append:\ Append(s_0, (Append(s_1, ...(Append(s_n, s_{n+1})...s_{m-1}, s_m)$ will have the set of known subterms to trigger exponentially for each bracketing which will lead to several instances with same information learnt. In order to avoid this, we introduce a new stub function $Append'(Seq\ T, Seq\ T) : Seq\ T.$ and modify our trigger by introducing a new function symbol in such a way that we lock further more instantiations.

$$\forall\ s_0,\ s_1 : Seq\ T\ \ [Append(s_0, s_1)]$$
$$Append(s_0, s_1) \approx Append'(s_0, s_1)$$

$$\forall\ s_0,\ s_1,\ s_2 : Seq\ T\ \ [Append(s_0, Append'(s_1, s_2))]$$
$$Append(s_0, Append'(s_1, s_2)) \approx Append'(Append(s_0, s_1), s_2)$$

Note that the new symbol introduced and the axiom is just a symbol equality which does not affect other axioms. In the second axiom, when the $Known$ terms will be learned with the ground terms or other active terms, the solver will also look for several possible combination of sub-terms. But this time, with the help of the new symbol introduced, we filter the duplicate instantiation of nested trigger with "stub" function. We have significantly limited the no. of subeterms to be triggered, as the new triggers will introduce inactive terms $Append'(Append(s_0, s_1), s_2)$ which do not get triggered again; as there is no trigger of such term in the axiomatization.

The new "stub" function potentially acts as a halting or controlling function to avoid equational axioms getting triggered redundantly.

3.2 Completeness

We use Herbrand model in our formalization. The procedure to [10]design a complete axiomatization W is to give a general method for completing the world L in which W axiomatization is true into a model of the theory. It means that the term of L that are interpreted must be given a value in the world which may create new equalities between them. This reasoning is easier if the new triggers are not unlocked in W axiomatization. Therefore, in multi-sorted logic, it is generally enough to restrict

the triggers such that, every trigger l in the axiomatization and every subterm t of l of an interpreted sort, either t is a variable or t is top-level in l. In this way, the only triggers that can be unlocked are then those where l becomes true because of the new equalities, which are generally easier to reason about.

Model modulo T : A world L is said to be a model of a closed formula ψ whenever L is complete and $L \rhd_T \psi$. We call ψ satisfiable if it has a model.

Another important point, when a trigger is guarding a disjunction (a nested function), then triggers should not prevent us from deducing an element of the disjunction when others are false (or not active) .

For example : The axioms described in the sequence axiomatization in next Sequence section with nested triggers must be triggered in such a way that the rewriting is possible in both ways. For example in the theory of Sequence axiomatization (shown in next section) must have duplicate axiom so that there trigger come from each side of the equality. For example, axiom *Lengt_Drop*.

Axiom 2 (*Length_ Drop*).

$$\begin{gathered}\forall\ s : Seq\ T, n : int\ \ [Length(Drop(s,n))] \\ (n < 0 \implies Length(Drop(s,n)) \approx Length(s))\ \&\ \\ (0 \leq n\ \&\ n < Length(s) \implies Length(Drop(s,n)) \approx Length(s) - n)\ \&\ \\ (Length(s) \leq n \implies Length(Drop(s,n)) \approx 0)\end{gathered}$$

Axiom 3 (*Length_ Drop_ Inv*).

$$\begin{gathered}\forall\ s : Seq\ T, n : int\ \ [Length(s), Drop(s,n)] \\ (n < 0 \implies Length(Drop(s,n)) \approx Length(s))\ \&\ \\ (0 \leq n\ \&\ n < Length(s) \implies Length(Drop(s,n)) \approx Length(s) - n)\ \&\ \\ (Length(s) \leq n \implies Length(Drop(s,n)) \approx 0)\end{gathered}$$

In this case if the above axiom was not added, we would not be able to get the conflict for the $\{length(s_1) \not\approx length(s_2), Drop(s_1,n) \approx Drop(s_2,n)\}$.

The approach to get completeness can also be extended to more general notion of Psi-Local Theory extensions. It is experienced that sometimes the local instances of axioms are not enough to decide the satisfiability modulo of the theory axiomatization. This is basically because in the First Order Logic with triggers (FOL*) the cut rule is not admissible in it. Therefore, we need to compute a stronger set $\psi(st(G))$ for the set $st(G)$ such that we have all the terms to compute the satisfiability.

$$\psi(st(G)) = st(G) \bigcup \{g(f(t) | t \in st(G)\}$$

The above formula states, the general intermediate step for computing stronger set of ground terms. The g and f are function symbols of same return type. To check the satisfiability of a set of ground terms G, it is sufficient to instantiate the axioms such that all term appear in $\psi(st(G))$.

Unfortunately, there can be no general way of computing this stronger set. Its also theory dependent. Although, it seems possible to specify Psi function in a symbolic way but as the solvers are today, this still needs to be done externally.

This could be also seen as an intermediate lemma which is required to prove the satisfiability. And it eventually helps us to avoid the restriction of not admitting the cut rule.

We can also introduce these stronger set of ground terms by triggering on all possible nested terms which we have with the help of the Psi function.

4 Sequences

We have used the above framework for Sequence axiomatization and have implemented successfully in our verification infrastructure VIPER. We also prove termination and Completeness for the sequence axiomatization and thus show that the triggered axiomatization can handle complex theories efficiently. Our tests, when the theory is heavily used, gives a better performance to the solver on the goals that comes from the verification of programs using this data-structure.

4.1 Presentation of the theory

Viper tool aims to provide automatic support for polymorphic sequences. The theory describes several functions over sequences. The choice of functions and the basis of axiomatization comes from the Dafny verification tool.

Let T be a polymorphic type. We define the sequence $Seq\,T$ ordered and of finite length over T. We first define function symbol $Empty() : Seq\,T$ which returns an empty sequence, it contains no elements.

Then we define a concatenation operation, which we call *Append*. To append a sequence means to add the elements of the second sequence at the end of first sequence. The constant function *Empty* and concatenation function *Append* satisfy the axioms for monoids. That is, *Empty* is an identity of *Append* and *Append* is also associative. We axiomatize this is in our axioms described later.

Furthermore, we define sequence extraction operation with functions *Take* and *Drop*. The *Take* function gives the first n elements of the sequence. The *Drop* function gives a sequence with its first n elements dropped.

We extend the theory to use integers. We define the function *Length* for sequences which is the number of elements. In order to refer to the elements directly as a reference we also define function *Index*. And thus in our axiomatization, we will add the knowledge of the few properties of the integers based on *length* and *index*.

The function *Contains* is based on the notion of membership. It is described by using the function symbol *Contains* whose value is true if only if the sequence s contains the element t.

4.2 Description of the Sequence Axiomatization

SMT solvers do not have a built-in support for Sequences. Several verification tools use first order axiomatization to support sequence theory automatically. We investigate further in detail on the first order axiomatization of sequences theory by keeping several guidelines and techniques to get a well reasoned trigger based axiomatization. We also give the detailed perspective of how the axiomatization can be classified and reasoned for termination. We also show where the prover will fail to instantiate the quantified axioms leading to incompleteness.

To start with axiomatization, computing a stronger set of $st(\psi)$ ground terms by saturating all possible combination of terms with triggers is useful. As mentioned earlier in the limitation of E-matching algorithm, the prover needs to have a term showing up in the ground terms to trigger some of the axioms. Therefore, several axioms are needed which can be seen as a special case of the extensionality axiom to trigger the equality. They basically help us to introduce the equal terms which also acts as an intermediate lemma. In general, its observed that one can take an idea of all these terms by constructing a larger set of ground terms. This further helps us to get a complete axiomatization.

The axioms are described below. They are classified according to their triggers.

Firstly, we have the *Length* term and all possible subterms triggers such as $Length(Empty())$, $Length(Singleton)$, $(Length(Append(s_0, s1)$, $Length(Take(s, n))$ and $Length(Drop(s, n))$.

Axiom 4 (*Length_ Positive*).

$$\begin{gathered}\forall\ s : Seq\ T\ \ [Length(s)] \\ 0 \leq Length(s)\end{gathered}$$

Axiom 5 (*Length_ Empty* 1).

$$Length(Empty()) \approx 0$$

Axiom 6 (*Length_ Empty* 2).

$$\begin{gathered}\forall\ s : Seq\ T\ \ [Length(s)] \\ Length(s) \approx 0 \implies s \approx Empty()\end{gathered}$$

Axiom 7 (*Length_ Singleton*).

$$\begin{gathered}\forall\ t : T\ \ [Length(Singleton(t))] \\ Length(Singleton(t)) \approx 1\end{gathered}$$

Axiom 8 (*Length_ Append*).

$$\begin{gathered}\forall\ s_0 : Seq\ T, s_1 : Seq\ T\ \ [Length(Append(s_0, s_1))] \\ Length(Append(s_0, s_1)) \approx Length(s_0) + Length(s_1)\end{gathered}$$

Axiom 9 (*Length_ Drop*).

$$\begin{gathered}\forall\ s : Seq\ T, n : int\ \ [Length(Drop(s, n))] \\ (n < 0 \implies Length(Drop(s, n)) \approx Length(s))\ \&\ \\ (0 \leq n\ \&\ n < Length(s) \implies Length(Drop(s, n)) \approx Length(s) - n)\ \&\ \\ (Length(s) \leq n \implies Length(Drop(s, n)) \approx 0)\end{gathered}$$

Axiom 10 (*Length_ Take*).

$$\begin{gathered}\forall\ s : Seq\ T, n : int\ \ [Length(Take(s, n))] \\ (n < 0 \implies Length(Take(s, n)) \approx 0)\ \&\ \\ (0 \leq n\ \&\ n < Length(s) \implies Length(Take(s, n)) \approx n)\ \&\ \\ (Length(s) \leq n \implies Length(Take(s, i)) \approx Length(s))\end{gathered}$$

We axiomatize the properties of Index term and all its sub-terms with

Axiom 11 (*Index_ Singleton*).

$$\forall\ t : T\ \ [Index(Singleton(t), 0)]$$
$$Index(Singleton(t),\ 0) \approx t)$$

Axiom 12 (*Index_ Append*).

$$\forall\ s_0 : Seq\ T, s_1 : Seq\ T, n : int\ \ [Index(Append(s_0, s_1), n)]$$
$$(n < Length(s_0) \implies Index(Append(s_0, s_1), n) \approx Index(s_0, n))\ \&$$
$$(Length(s_0) \leq n \implies Index(Append(s_0, s_1), n) \approx Index(s_1, n - Length(s_0)))$$

Axiom 13 (*Index_ Take*).

$$\forall\ s : Seq\ T, n : int, j : int\ \ [Index(\ Take(s, n), j)]$$
$$0 \leq j\ \&\ j < n\ \&\ j < Length(s) \implies Index(\ Take(s, n), j) \approx Index(s, j)$$

Axiom 14 (*Index_ Drop*).

$$\forall\ s : Seq\ T, n : int, j : int\ \ [Index(Drop(s, n), j)]$$
$$0 \leq n\ \&\ 0 \leq j\ \&\ j < Length(s) - n \implies Index(Drop(s, n), j) \approx Index(s, j + n)$$

We axiomatize *Contains* terms and its sub-terms.

Axiom 15 (*Contains*).

$$\forall\ s : Seq\ T, x : T\ \ [Contains(s,x)]$$
$$Contains(s,x) \iff (\exists i : int :: [Index(s, i)] 0 \leq i\ \&\ i < Length(s)\ \&\ Index(s, i) \approx x)$$

Axiom 16 (*Contains_ Empty*).

$$\forall\ x : T\ \ [Contains(Empty(), x)]$$
$$!Contains(Empty(),\ x))$$

Axiom 17 (*Contains_ Append*).

$$\forall\ s_0 : Seq\ T, s_1 : Seq\ T, x : T\ \ [Contains(Append(s_0, s_1), x)]$$
$$Contains(Append(s_0, s_1), x) \iff Contains(s_0, x) || Contains(s_1, x)$$

Axiom 18 (*Contains_ Take*).

$$\forall\ s : Seq\ T, n : int, x : T\ \ [Contains(\ Take(s, n), x)]$$
$$Contains(Take(s,\ n),\ x)$$
$$\iff (\exists\ i : int\ \ [Index(s, i)]\ \ 0 \leq i\ \&\ i < n\ \&\ i < Length(s)\ \&\ Index(s, i) \approx x)$$

Axiom 19 (*Contains_ Drop*).

$$\forall\ s : Seq\ T, n : int, x : T\ \ [Contains(Drop(s, n), x)]$$
$$Contains(Drop(s,\ n),\ x)$$
$$\iff (\exists\ i : int\ \ [Index(s, i)]\ \ 0 \leq n\ \&\ n \leq i\ \&\ i < Length(s)\ \&\ Index(s, i) \approx x)$$

Axiom 20 (*Contains_ Singleton*).

$$\forall\ x, y : T\ \ [Contains(Singleton(x), y)]$$
$$Contains(Singleton(x),y) \iff x \approx y$$

We define the associativity property of *Append* function with the help of a recursive trigger and performance optimization as described in general guidelines with the help of auxillary *Append'* function

Axiom 21 (*Empty_ Append* 1).

$$\forall\ s : Seq\ T\ \ [Append(Empty(), s)]$$
$$Append(Empty(),s) \approx s$$

Axiom 22 (*Append_ Empty* 2).

$$\forall\ s : Seq\ T\ \ [Append(s, Empty()]$$
$$Append(s,Empty()) \approx s$$

Axiom 23 (*Append*).

$$\forall\ s_0 : Seq\ T, s_1 : Seq\ T\ \ [Append(s_0, s_1)]$$
$$Append(s_0, s_1) \approx Append'(s_0, s_1)$$

Axiom 24 (*Append_ Append'*).

$$\forall\ s_0 : Seq\ T, s_1 : Seq\ T, s_2 : Seq\ T\ \ [Append(s_0, Append'(s_1, s_2))]$$
$$Append(s_0, Append'(s_1, s_2)) \approx Append'(Append(s_0, s_1), s_2)$$

Take and Drop additive properties are defined with recursive triggering function with the help auxillary functions *Take'* and *Drop'* respectively to avoid exponential blow up.

Axiom 25 (*Drop_ base*).

$$\forall\ s : Seq\ T, n : int\ \ [Drop(s, n)]$$
$$n == 0 \implies Drop(s, n) \approx s$$

Axiom 26 (*Take_ base*).

$$\forall\ s : Seq\ T, n : int\ \ [Take(s, n)]$$
$$n == 0 \implies Take(s, n) \approx Empty()$$

Axiom 27 (*Drop_ Recursive*).

$$\forall\ s : Seq\ T, m, n : int\ \ [Drop, n]$$
$$0 \leq n\ \&\ n \leq Length(s) \implies Drop(s, n) \approx Drop'(s, n)$$

Axiom 28 (*Drop_ Drop'_ Recursive*).

$$\forall\ s : Seq\ T, m, n : int\ \ [Drop(Drop'(s, m), n)]$$
$$0 \leq m\ \&\ 0 \leq n\ \&\ m + n \leq Length(s) \implies Drop(Drop'(s, m), n) \approx Drop'(s, m + n)$$

Axiom 29 (*Take_ Recursive*).

$$\forall\ s : Seq\ T, m, n : int\ \ [Take(s, n]$$
$$0 \leq n\ \&\ n \leq Length(s)\ \implies\ Take(s, n) \approx Take'(s, n)$$

Axiom 30 (*Take_ Take'_ Recursive*).

$$\forall\ s : Seq\ T, m, n : int\ \ [Take(Take'(s, m), n)]$$
$$0 \leq m\ \&\ 0 \leq n\ \&\ m + n \leq Length(s)\ \implies\ Take(Take'(s, m), n) \approx Take'(s, m + n)$$

Theorem 4.1. The axiomatization is terminating, sound and complete with respect to the same axiomatization without triggers and witnesses on verification conditions coming from our intermediate verification language *Silver* in VIPER verification infrastructure.

4.3 Termination

We show that our verification conditions coming from VIPER programs which uses sequences axiomatization is terminating.

We use the fact that the triggers only contain uninterpreted function symbols. We only need to show that considering the set of terms generated via E-matching with the above axioms and any finite set of initial terms and equalities, the instantiation will terminate. Indeed, as mentioned in general criteria, the underlying theory T cannot create a new terms starting with every uninterpreted function symbol, hence there can only be a finite number of instances generated by our axiomatization.

Length Terms (α)	Index Terms (β)	Contains Terms (γ)
$Length(Singleton(t), c)$	$Index(Singleton(t), c)$	$Contains(Singleton(t), 0)$
$Length(Append(s_0, s_1)$	$Index(Append(s_0, s_1)$	$Contains(Append(s_0, s_1), e)$
$Length(Take(s, n))$	$Index(Take(s, n), i)$	$Contains(Take(s, n), e)$
$Length(Drop(s, n))$	$Index(Drop(s, n), j)$	$Contains(Drop(s, n), e)$

We classify the axioms and their triggers in such a way that they can be reasoned independently or reasoned relatively to other classified sets. To do this, we separate the triggering terms. By triggering terms we mean the set of terms that can be triggered when the terms are known. In sequence axiomatization we do this classification according to *Length*, *Index* and *Contains* as their sub-terms.

We first consider $Length(s)$ trigger from the axiom *Length_ Positive* and *Length_ Empty*. The *Length_ Positive* axiom does not create any new triggering term except constants. The *Length_ Empty* axiom only creates a constant logical function *Empty*() which of course does not trigger further.

Now, we look for all the length terms with the sub-terms as shown above. We see all the axioms *Length_ Singleton*, *Length_ Append*, *Length_ Take*, *Length_ Drop* can unlock only length term triggers. We can see this as a rewriting of equational axioms from (t_{lhs} to t_{rhs}). Now, let us take this set of the triggering *Length* terms to be α. This set α can expand with the instantiation of triggering *Length* terms in the presence of ground terms or known terms. In order to prove the termination it is necessary to argue on the fact that this set will stop expanding eventually when all the terms are saturated or

partial model is built. We know that we start with finite initial terms, therefore the only possible way to see the non-termination is when the set of *Length* axioms are diverging. Now, we analyse our axioms carefully and we find that all the *Length* terms when triggered unlocks "smaller" terms which do not introduce new terms, converging to the end of the finite terms we started with. Therefore, according to the axioms the *length* terms will only unlock converging triggering *length* terms, and we see the termination eventually.

Similarly we look for *Index* terms. The function *Index* terms does get triggered in the *Contains* axiom and in the existential quantification. If we look at the axioms of all *Index* term, we see that this can only lead to new triggering terms of *Length* or *Index*. The *Length* triggering terms will saturate to the set α and follow the instantiation as explained before. And then, we further look with the sub-terms triggered with *Index* axioms. Now, let us take another set β for the triggering *index* terms. Similar to the *Length* axioms, we analyse our axioms and find all the triggers which unlock only converging triggering *Index* terms. Therefore, even if the set β expands with ground and known terms but due to converging triggering with finite terms to start with, it will terminate eventually.

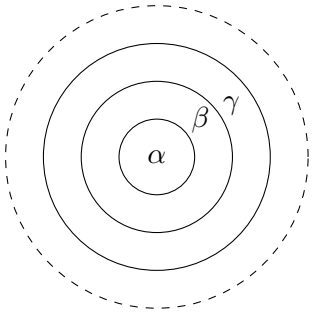

Figure 1: Representation of set α, β and γ

Now, we consider *Contains* terms. Let us take the set γ for the triggering contains terms. As the trigger *Contains* saturates directly into *Index* and *Length* terms and thus respectively into the set β and α which then follows the above argument. We further see the *Contains* and its sub-terms in the axioms *Contains_ Singleton*, *Contains_ Append*, *Contains_ Take* and *Contains_ Drop* has only converging triggering terms . This terms will be saturated only to the terms *Contains* and *Index* and *Length*, thus into the set γ, β, and α respectively which will expand with the ground terms and known terms, but eventually gives a finite number of instantiation of axioms.

For the recursively triggered axioms *Append_ Append*, *Take_ Take* and *Drop_ Drop* we keep the base axiom and the recursive axiom. The application of these two axioms are eventually stopped when the conditions of the base axioms are satisfied.

We do believe that the sets α, β and γ will expand with the no. of terms getting instantiated, However this expansion will eventually terminate because we start with only finite uninterpreted terms. we cannot have an infinite triggering unless we have a mutually recursive trigger or a matching loop. We avoid the possibility of matching loop in our axiomatization by triggering only towards one side. If we abstract the triggering axiomatization, we can see it as a triggering in only only one direction hierarchically. The *Contains* term triggers unlock only *Index* and *Length* term triggers. And the *Index* term triggers unlock only to the *Length* triggers. The *Length* triggers unlock only the *Length* triggers.

Therefore, with above arguments we conclude these axioms will instantiate triggers finitely and thus can be added to the solver without compromising the termination of it.

4.4 Completeness

Firstly we note that the axiomatization is not complete in general with the extensionality of sequences.

Axiom 31 (*Sequence_ Equal*).

$$\forall\ s_0 : Seq\ T, s_1 : Seq\ T\ \ [Equal(s0, s1)]$$
$$Equal(s_0, s_1) \Longleftrightarrow Length(s_0) \approx Length(s_1) \& \forall\ j : int\ \ [Index(s_0, j)][Index(s_1, j)]$$
$$0 \leq j \& j < Length(s_0)\ \Longrightarrow\ Index(s_0, j) \approx Index(s_1, j))$$

Axiom 32 (Extensionality).

$$\forall\ a : Seq\ T, b : Seq\ T\ \ [Equal(a, b)]$$
$$Equal(a,b)\ \Longrightarrow\ a \approx b)$$

One can come up easily with a counter-example to show the axiomatization is incomplete if not *Equal* term shows up in the literals. Indeed such a case can only happen in practice, if either there is a trigger containing twice the same sequence variable, in which case a new equality will allow the trigger to match with the extensionality or if there are two nested triggers with the same sequence variable. Generally, in practice, the first case happens rarely and if it does it is covered by the extensionality. For the second case, as mentioned in the section 3.2 designing complete axiomatization; it does happen with the nested triggers such as $Length(Drop(s, n))$, the axiomatization will be incomplete as it won't be able to find the unsatisfiability in $\{Length(s_1) \not\approx Length(s_2), Drop(s_1, n) \approx Drop(s_2, n)\}$. The second case, can be solved when added duplicate inverse axiom with multi-trigger.

We have handled the nested recursive applications with recursive triggers. We have three recursive triggers $Append(Append(s_1, s_2), s_s)$, $Drop(Drop(s, i), j)$ and $Take(Take(s, i), j)$. These triggers and axioms define properties when nested to themselves. They help to prove the sub-goals that may arrive from the verification conditions. For example, it would not have been possible to get the unsatisfiability in $\{drop(drop(s, i), j) \approx s_t, drop(s, i + j) \not\approx drop(s_t)\}$.

There are couple of axioms that are specialization of existensionality axioms.

Axiom 33 (*Append_ Drop*).

$$\forall\ s_0 : Seq\ T, s_1 : Seq\ T, n : int\ \ [Append(Drop(s_0, n), s_1)]$$
$$0 < Length(s_0)\ \&\ n \geq 0\ \&\ n \leq Length(s_0)\ \Longrightarrow$$
$$Append(Drop(s_0, n), s_1) \approx Drop(Append(s_0, s_1), n)))$$

Axiom 34 (*Append_ Take*).

$$\forall\ s_0 : Seq\ T, s_1 : Seq\ T, n : int\ \ [Append(s_0, Take(s_1, n))]$$
$$0 < Length(s_1)\ \&\ n \leq 0\ \&\ n \leq Length(s_1)\ \Longrightarrow$$
$$Append(s_0, Take(s_1, n)) \approx Take(Append(s_0, s_1), n + Length(s_0)))$$

Basically, because in sequence axiomatization the solver could not learn the equality when Append and Drop or Append and Take function overlaps, and when they do not overlap. Therefore the axioms *Append_ Drop* and *Append_ Take* were required. These axioms will be theoretically redundant but are useful in the triggered axiomatization of sequences. Its because during the time of experimentation, we noticed, the solver wasn't able to apply the extensionality.

Figure 2: *Append_ Drop* and *Append_ Take*

As shown in the figure above, cut made by m and n would have certain part of the sequences still equal irrespective of the cut made before or after the *Append* operation.

In order to prove that we are complete except the case of extensionality, we first need a lemma that states the equalities between integers which can be safely added to the partial model of the axiomatization :

Lemma 4.2. [10] Let L be a world in which the axiomatization is true and t_1,$t_2 \in \mathcal{T}(L)$ be two type integers. If $L \nvdash t_1 \not\approx t_2$ then the axiomatization is also true in $L \cup t_1 \approx t_2$.

Let G be a set of literals and L a world in which G and the axiomatization are true. We construct a model from L for the axiomatization without triggers and witnesses. Since $L \rhd_T G$, it is also a model of G.

Since L is a *T-satisfiable*, let I_T be a model of L. No integer constant appears in a trigger of the axiomatization. As a consequence, the axiomatization is true in $L \cup \{i \approx i | i \text{ is an integer constant}\}$. For every term $t \in \mathcal{T}(L)$ of the form *Length(s)*, we add $t \approx I_T(t)$ to L. By Lemma, the axiomatization is still true in L.

For every term s of type *Seq* in L, if *Length*(s) is not in $\mathcal{T}(L)$ modulo T, we add *Length*$(co) \approx 0$ to L and, for every index i of type *int* if *Index*(s, i) is not in L, we add a *Index*$(s, i) \approx 0$. This decides that the sequences that are not forced to be non-empty are empty and index that are not forced to be valid are invaild in Sequence s. The axiomatization will be still true in L. Indeed, thanks to the axiom *Length_ Positive* which say *Length*(s) is in $\mathcal{T}(L)$ whenever there is any *Index*(s, i) for sequence s is in $\mathcal{T}(L)$.

Now, we give a value to *Contains*(s, v) for each known term t of Type T and each known term s of *Seq* T in L. For every s and x such that $L \nvdash_T$ *Contains*$(s, v) \approx t$, we add the literal *Contains*$(s, v) \not\approx t$ to L. Since *Contains* is uninterpreted, the set of literals L is still satisfiable in T. The axiomatization will be still true in L. Since axioms guarded with *Contains*(s, v) term contains a negative occurrence of *Contains*$(s, v) \approx t$. As a consequence, they are all automatically satisfied when assuming *Contains*$(s, v) \not\approx t$.

Finally, we handle *Empty*. For every known term s of type $Seq.1cmT$ in L, if s is not equal to *Empty* then add *Empty*$(s) \not\approx t$ to L. If the length of the seq s is 0 we already have $L \vdash_T s \approx$ *Empty*(). Thanks to the *Length*(*Empty*) axiom and as a consequence the axiomatization will be still true in L.

4.5 Soundness

We show that, if a set of literal G has a model in the axiomatization without triggers and witnesses then there is a total model of G and the axiomatization [10]. If I is the model of a set of literals G in

the axiomatization without triggers and witnesses, we define $L = l|I(l) = T$. By construction of L, L is a total model of G. Since L is total, for every axiom ψ of the axiomatization, $L \rhd_T \psi$.

5 Related Work

Besides first order axiomatization, there are several ways that can be used to support new theories in SMT solvers. As described in this paper by Bjorner [3], several non-native theories can be supported using Z3. This paper also presents an API which can be used to add decision procedures. Both the first order axiomatization and encoding to an already supported theories through API, requires a lot of thinking to get a coherent design and manual proofs of completeness and termination. We find each of these techniques can be better than others specific to the theories.

There are various instantiation techniques to support quantifier in First order logic other than the trigger-based instantiation. Model-based quantifier instantiation [12] is one of the commonly used technique for generating instances. Generally, its not considered well performing but on the other hand its accurate, since it allows to continue the search when otherwise the solver would have stopped with the partial model.

Simplify is one of the first legacy solver, the first one to have efficiently combine theories with quantifiers reasoning using E-matching algorithm [9]. They define several triggering techniques in their technical report. Most of the solvers have heuristics to select the triggers automatically in the E-matching algorithm. However, its commonly agreed that manual triggers are sometimes more efficient to generate better instances [16]. A Lot of work has also been done on defining an efficient mechanism for finding the instances with triggers.

In [11] paper, they give a generic technique and guidelines that can be followed to do an efficient trigger based axiomatization which works as a custom decision procedure. We use the same formalizataion and extend the general guidelines and techniques. Recently, in this paper [1] they also show how to obtain a complete decision procedure for local theory extensions via E-matching, an important class of theory that are decidable using a finite instantiation of axioms.

Several theories has been supported as a decision procedure efficiently using these techniques. In Claire Dross thesis [10] , she has given a formal trigger-based axiomatization with proofs for sets and linked list. We provide a formalization for the sequences theory in this thesis which is most widely used in various verification infrastructure.

6 Experimentation and Debugging

We evaluated our techniques on a set of benchmarks of silver intermediate verification language in the VIPER verification tool [14] . The benchmarks relies on the automatic support for sets and sequences theories. The intermediate verification language Silver which supports specification in first-order logic and separation logic. The tool reduces the program and specification into verification conditions modulo several other theories such as sets and sequences.

We can see the heirarchy of tool as follows:

- Base theory: At the lowest level we have UFLIA
- Sets and Sequence

- Implicit dynamice frames and separation logic (permission logic)
- program specific extensions

The programs considered were general sorting algorithms, common data structure operations and abstract datatypes, linked lists etc.

The initial set of sequence axiomatization based on Dafny verification tool were not complete and sometimes had several unpredictable behaviour due to diverging triggers. After the formalization and new set of axioms we could fix them and also could improve the overall performance by restricting duplicate instantiations. The VIPER test suite performance improved from 250 sec to 155 sec. We were also able to automatically verify the sequences and set specific assertions. The similar behaviour is also expected with the Dafny verification tool.

The particular example of "two way boolean recursive sort" which makes heavy use of sequences verified in 40 sec with the initial set of axioms, whereas after with new set of axiomatization it took less than 15 sec. The similar performance improvement was seen in Silicon the symbolic execution tool at the back-end of Viper (from 15 sec to 5 sec).

During experimentation and debugging we used the debugging tool Axiom profiler from Vcc verification tool [5]. Sometimes two triggers from different axioms behaves in a very unpredicatable manner and its hard to detect them. Axiom profiler helped us to see the possible matching loop in these cases. One of the example where we could debug a diverging trigger was in the axiom *Append_Drop* and *Drop_Drop_Recursive*. The triggering diverged when triggered on $Drop(Append(s_0, s_1), n)$ instead of $Append(Drop(s_0, n), s_1)$.

7 Conclusion

In this thesis, we have investigated and proposed trigger-based axiomatization techniques with respect to termination, completeness and soundness. Keeping these techniques in mind, we worked concretely on sequence axiomatization; proved and argued about its soundness, termination and completeness. We also presented a technique to improve the performance issue of the solver. The result of these gives a significant improvement over the previous version of the axiomatization.

Related to the sequences axiomatization, one could work further more to provide a complete support to the extensionality. There are several other related problems which are not addressed here, such as various instantiation techniques : lazy, eager and hybrid instantiations. These techniques are used for generating instantiation tree and can have different behaviour for different triggger-based axiomatizatons. It has been shown in [18], sometimes switching to one or another can help the solver to find conflicting instances faster. It will be interesting to investigate furthermore about triggering strategies in the perspective of how the instantiation tree is generated.

References

[1] Kshitij Bansal, Andrew Reynolds, Tim King, Clark Barrett, and Thomas Wies. Deciding local theory extensions via E-matching. In *Proceedings of the 27 International Conference on Computer Aided Verification (CAV '15)*. Springer, July 2015. San Francisco, CA.

[2] Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. Cvc4. In *Proceedings of the 23rd International Conference on Computer Aided Verification*, CAV'11, pages 171–177, Berlin, Heidelberg, 2011. Springer-Verlag.

[3] Nikolaj Bjørner. Engineering theories with z3. In *Proceedings of the 9th Asian Conference on Programming Languages and Systems*, APLAS'11, pages 4–16, Berlin, Heidelberg, 2011. Springer-Verlag.

[4] François Bobot, Jean christophe Filliâtre, Claude Marché, and Andrei Paskevich. Why3: Shepherd your herd of provers. In *In Workshop on Intermediate Veri cation Languages*, 2011.

[5] Ernie Cohen, Markus Dahlweid, Mark A. Hillebrand, Dirk Leinenbach, Michal Moskal, Thomas Santen, Wolfram Schulte, and Stephan Tobies. Vcc: A practical system for verifying concurrent c. In *Theorem Proving in Higher Order Logics, 22nd International Conference, TPHOLs 2009*, volume 5674 of *Lecture Notes in Computer Science*, pages 23–42. Springer, 2009.

[6] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, July 1962.

[7] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'08/ETAPS'08, pages 337–340, Berlin, Heidelberg, 2008. Springer-Verlag.

[8] Leonardo de Moura and Nikolaj Bjørner. Efficient e-matching for smt solvers. Technical report, 2007.

[9] David Detlefs, Greg Nelson, and James B. Saxe. Simplify: A theorem prover for program checking. *J. ACM*, 52(3):365–473, May 2005.

[10] Claire Dross. *Generic Decision Procedures for Axiomatic First-Order Theories*. Thèse de doctorat, Université Paris-Sud, April 2014.

[11] Claire Dross, Sylvain Conchon, Johannes Kanig, and Andrei Paskevich. Reasoning with triggers. In Pascal Fontaine and Amit Goel, editors, *SMT 2012*, volume 20 of *EPiC Series*, pages 22–31. EasyChair, 2013.

[12] Yeting Ge and Leonardo Moura. Complete instantiation for quantified formulas in satisfiabiliby modulo theories. In *Proceedings of the 21st International Conference on Computer Aided Verification*, CAV '09, pages 306–320, Berlin, Heidelberg, 2009. Springer-Verlag.

[13] Carsten Ihlemann, Swen Jacobs, and Viorica Sofronie-Stokkermans. On local reasoning in verification. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'08/ETAPS'08, pages 265–281, Berlin, Heidelberg, 2008. Springer-Verlag.

[14] U. Juhasz, I. T. Kassios, P. Müller, M. Novacek, M. Schwerhoff, and A. J. Summers. Viper: A verification infrastructure for permission-based reasoning. Technical report, ETH Zurich, 2014.

[15] K. Rustan M. Leino. Dafny: An automatic program verifier for functional correctness. In *Proceedings of the 16th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, LPAR'10, pages 348–370, Berlin, Heidelberg, 2010. Springer-Verlag.

[16] MichałMoskal. Programming with triggers. In *Proceedings of the 7th International Workshop on Satisfiability Modulo Theories*, SMT '09, pages 20–29, New York, NY, USA, 2009. ACM.

[17] MichałMoskal, Jakub ński, and Joseph R. Kiniry. E-matching for fun and profit. *Electron. Notes Theor. Comput. Sci.*, 198(2):19–35, May 2008.

[18] Andrew Reynolds, Cesare Tinelli, and Leonardo de Moura. Finding conflicting instances of quantified formulas in smt. In *Proceedings of the 14th Conference on Formal Methods in Computer-Aided Design*, FMCAD '14, pages 31:195–31:202, Austin, TX, 2014. FMCAD Inc.

[19] Alexandre Riazanov and Andrei Voronkov. *Vampire 1.1*, volume 2083 of *Lecture Notes in Computer Science*, chapter 29, pages 376–380. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

A Figures

The figure shown below describes which trigger in Sequence axiomatization unlocks which triggering term. The directed acyclic graph constructed helps us to see if there is any possible cycle which can be a mutually recursive trigger (matching loop). However, one has to still do a manual proof because there can be a possibility of having diverging triggers which cannot be visualized in the Directed graph as cycles.

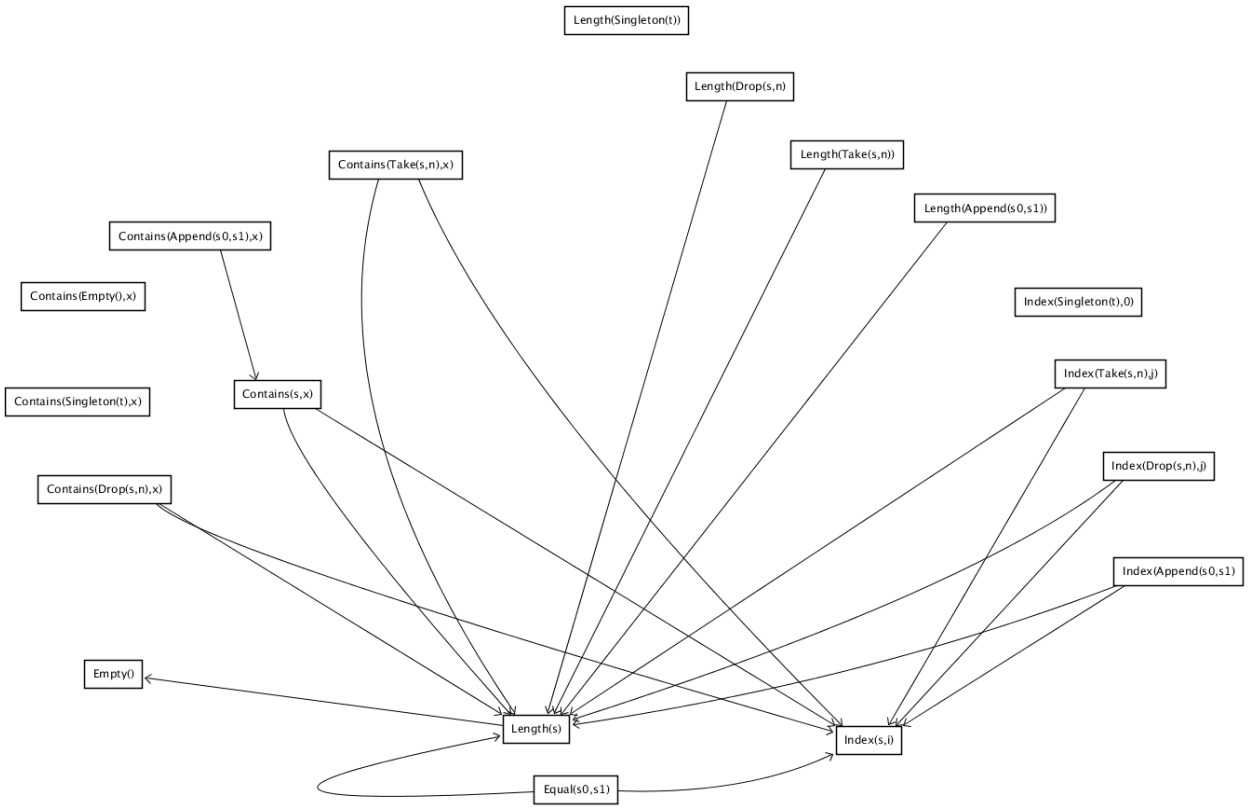

Figure 3: Directed graph of triggers in sequence axiomatization

B Making extensionality stronger

The below axioms on the extensionality were also found useful in performance improvement. They helped the prover to add known terms and thus making the set of Ground terms more stronger, which avoids instantiating new instances.

Axiom 35 (Extensionality_reflexive).

$$\forall\ a : Seq\,T, b : Seq\,T\ \ [Equal(a, b)]$$

$$Equal(a,b) \implies Equal(b,a)$$

Axiom 36 (Extensionality_transitive).

$$\forall\ a : Seq\ T, b : Seq\ T, c : Seq\ T\ \ [Equal(a,b), Equal(b,c)]$$
$$Equal(a,b)\ \ \&\ \ Equal(b,c) \implies Equal(a,c)$$

Axiom 37 (Extensionality_transitive 1).

$$\forall\ a : Seq\ T, b : Seq\ T, c : Seq\ T\ \ [Equal(a,b), Equal(a,c)]$$
$$Equal(a,b)\ \ \&\ \ Equal(b,c) \implies Equal(a,c)$$